MUL Source

- Perform unsigned multiply operation.
- This instruction assumes one of the operand in AL or AX.
- Source can be a register or memory location.
- The multiplication instruction contains one operand because it always multiplies the operand times the contents of register AL.
- If source operand is a byte, **AX = AL * Source**
- If source operand is a word, **(DX AX) = AX * Source**
- Source operands can not be an immediate data
- It modifies CF and OF (AF,PF,SF,ZF undefined).

Format: MUL S

Operation: (AX) ← (AL) * (S8)

Operation: (DX-AX) ← (AX) * (S16)

→ After MUL

→ CF/OF = 0, If the upper half of the result is zero.

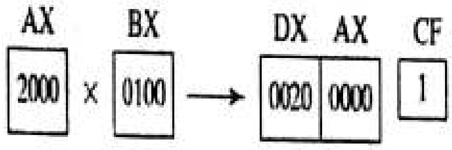
→ CF/OF = 1, otherwise.

Assembly Language	*Operation*
MUL CL	AL is multiplied by CL; the unsigned product is in AX
IMUL DH	AL is multiplied by DH; the signed product is in AX
IMUL BYTE PTR[BX]	AL is multiplied by the byte contents of the data segment memory location addressed by BX; the signed product is in AX
MUL CX	AX is multiplied by CX; the unsigned product is in DX–AX
IMUL DI	AX is multiplied by DI; the signed product is in DX–AX
MUL WORD PTR[SI]	AX is multiplied by the word contents of the data segment memory location addressed by SI; the unsigned product is in DX–AX

```
Dec       Hex
 61        3D
x 90      x 5A
5490       262     A x D = 82,  A x 3 = 1E + 8 = 26
          131      5 x D = 41,  5 x 3 = F + 4 = 13
          1572₁₆ = 5490₁₀
```

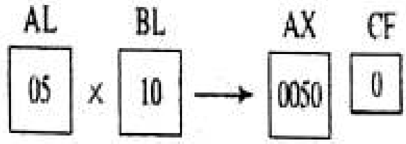
❖ 16-Bit Multiplication

```
MOV AX, 2000H    ; a word is moved to AX
MOV BX, 0100H    ; immediate data must be in
                   BX register
MUL BX           ; DX:AX = 00200000H, CF = 1
```



❖ 8-Bit Multiplication

```
MOV AL, 5H       ; a byte is moved to AL
MOV BL, 10H      ; immediate data must be in
                   BL register
MUL BL           ; AX = 0050h, CF = 0
```



Examples:

→ (AL) = 80h, (BL) = FFh
After MUL BL
(AH) = 7Fh, (AL) = 80h, CF/OF = 1.

→ (AX) = 0001h, (BX) = FFFFh.
After MUL BX
(DX) = 0000, (AX) = FFFFh, CF/OF = 0.

→ (AX) = FFFFh, (BX) = FFFFh.
After MUL BX
(DX) = FFFE, (AX) = 0001h, CF/OF = 1.

→ (AX) = 0FFFh, (BX) = 0FFFh.
After MUL BX
(DX) = 00FFh, (AX) = E001h, CF/OF = 1.

→ (AX) = 0100h, (BX) = FFFFh.
After MUL BX
(DX) = 00FFh, (AX) = FF00h, CF/OF = 1.

2

byte x byte :

- One of the operands must be in AL. The other operand can be either in a register or in memory.
- After the multiplication the result is in AX.

word x word :

- One of the operands must be in AX . The other operand can be either in a register or in memory.
- After the multiplication the lower word is in AX and the higher word is in DX.

word x byte :

- Similar to word x word, but AL contains byte operand and AH must be zero.

❖ Summary of Multiplication of Unsigned Numbers

Multiplication	Operand 1	Operand 2	Result
byte x byte	AL	register or memory	AX
word x word	AX	register or memory	DX-AX
word x byte	AL=byte, AH=0	register or memory	DX-AX

❖ 16-Bit Multiplication

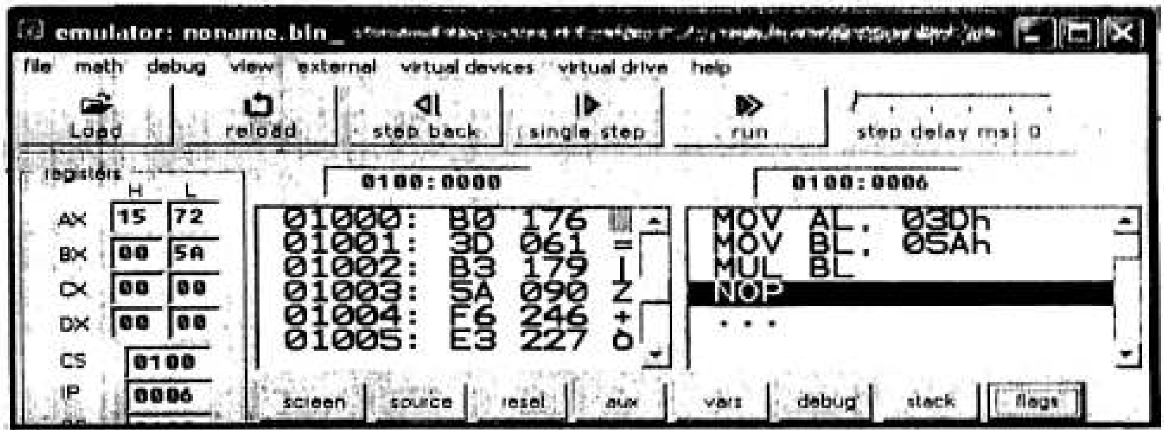
```
MOV CX, 2378H        ; a word is moved to CX
MOV BX, 2F79H        ; immediate data must be in
                       BX register
MOV AX, CX           ; position data
MUL BX               ; multiply
MOV DI, OFFSET       ; offset address
MOV [DI], AX         ; store AX in DI memory
                       location
MOV [DI+2], DX       ; store DX in DI+2 memory
                       location
```

❖ DX-AX = AX*CX after multiplication.

3

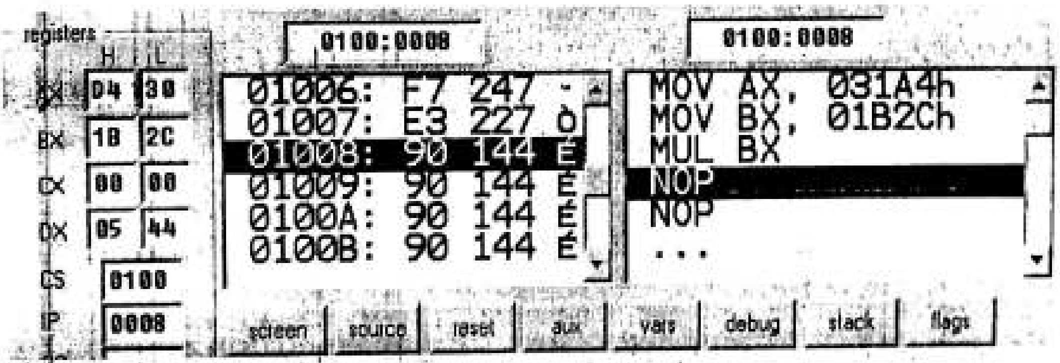
```
0000   B0 3D   MOV AL,3DH   ;AL=3DH
0002   B3 5A   MOV BL,5AH   ;BL=5AH
0004   F6 E3   MUL BL       ;AX=ALxBL
```

product = 1572H is in AX




```
0000   B8 A4 31   MOV AX,31A4H   ;AX=31A4H
0003   BB 2C 1B   MOV BX,1B2CH   ;BX=1B2CH
0006   F7 E3      MUL BX         ;DX:AX = AX x BX
```

```
     31A4
    x1B2C
   ------
    253B0
    6348
   2220C
   31A4
 --------
 0544D430
```



4

IMUL Source

- The IMUL (signed multiply) instruction performs signed integer multiplication.
- Unlike the MUL instruction, IMUL preserves the sign of the product. It does this by sign extending the highest bit of the lower half of the product into the upper bits of the product.
- If source operand is a byte, **AX = AL * Source**
- If source operand is a word, **(DX AX) = AX * Source**
- Source operands cannot be an immediate data
- It modifies CF and OF (AF,PF,SF,ZF undefined)

Signed Multiplication Summary:

Multiplication	Operand 1	Operand 2	Result
byte x byte	AL	register or memory	AX
word x word	AX	register or memory	DXAX
word x byte	AL = byte CBW	register or memory	DXAX

IMUL Instruction:

→ Multiply signed numbers

Format: **IMUL S**

IMUL reg/mem8 **;AX = AL*reg/mem8**

Operation: (AX) ← (AL) * (S8)

IMUL reg/mem16 **;DX:AX = AX * reg/mem16**

Operation: (DX-AX) ← (AX) * (S16)

→ After IMUL

→ CF/OF = 0, If the upper half of the result is the sign extension of the lower half (this means that the bits of the upper half are the same as the sign bit of the lower half)

→ CF/OF = 1, otherwise.

5

- The following instructions multiply AL by BL, storing the product in AX. Although the product is correct, AH is not a sign extension of AL, so the Overflow flag is set:

```
MOV AL, 48      ; a byte is moved to AL
MOV BL, 4       ; immediate data must be in
                  BL register
IMUL BL         ; AX = 00C0H, OF = 1
```

$48_d = 30H$

30H
4
C0

00 C0 last bit = 1

- The following instructions multiply 48 by 4, producing +192 in DX:AX. DX is a sign extension of AX, so the Overflow flag is clear:

```
MOV AX, 48      ; a word is moved to AX
MOV BX, 4       ; immediate data must be
                  in BX register
IMUL BX         ; DX:AX = 000000C0H,
                  OF = 0
```

✓

- The following instructions multiply-4 by 4, producing (-16) in AX. AH is a sign extension of AL so the Overflow flag is clear:

```
MOV AL, -4      ; a byte is moved to AL
MOV BL, 4       ; immediate data must
                  be in BL register
IMUL BL         ; AX = FFF0H, OF = 0
```


$-4 \times 4 = -16$

$16_d = 00010000 = 0010H$

FFF0 $= -16_d$

6

Examples:

→ (AL) = 80h, (BL) = FFh.
After IMUL BL
(AH) = 00h, (AL) = 80h, CF/OF = 1.

→ (AX) = 0001h, (BX) = FFFFh.
After IMUL BX
(DX) = FFFFh, (AX) = FFFFh, CF/OF = 0.

→ (AX) = FFFFh, (BX) = FFFFh.
After IMUL BX
(DX) = 0000, (AX) = 0001h, CF/OF = 0.

→ (AX) = 0FFFh, (BX) = 0FFFh.
After IMUL BX
(DX) = 00FFh, (AX) = E001h, CF/OF = 1.

→ (AX) = 0100h, (BX) = FFFFh.
After IMUL BX
(DX) = FFFFh, (AX) = FF00h, CF/OF = 0.

Dec	Hex
165	A5
x 36	x 24
990	294
495	14A
5940	1734

$5940_{10} = 1734_{16}$

But A5 = 10100101 can be a signed number
2's comp = 01011011 = 5BH = 91_{10}
Therefore, A5 can represent -91

Dec
−91
x 36
546
273
−3276

$3276_{10} = 0CCC_{16}$
= 0000 1100 1100 1100
2's comp = 1111 0011 0011 0100
= F334H

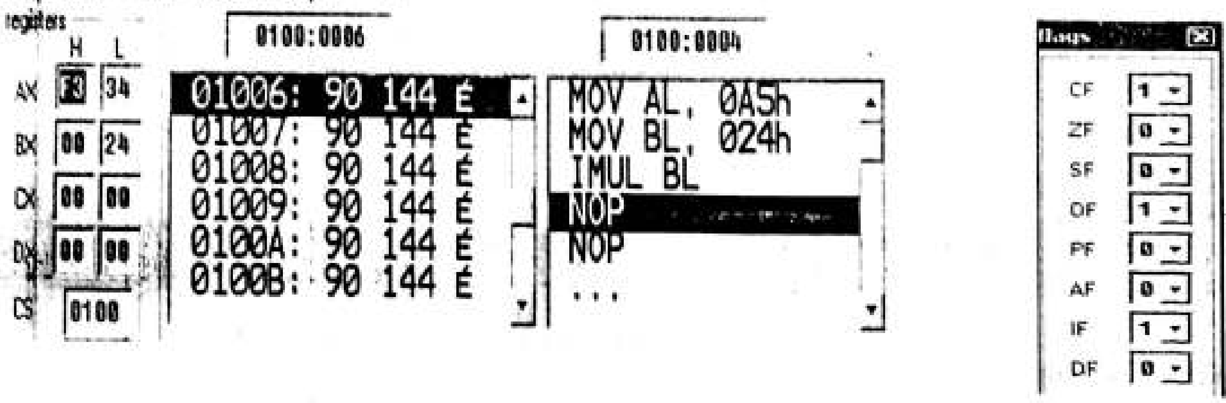
Therefore, for signed multiplication
A5H x 24H = F334H
and not 1734H

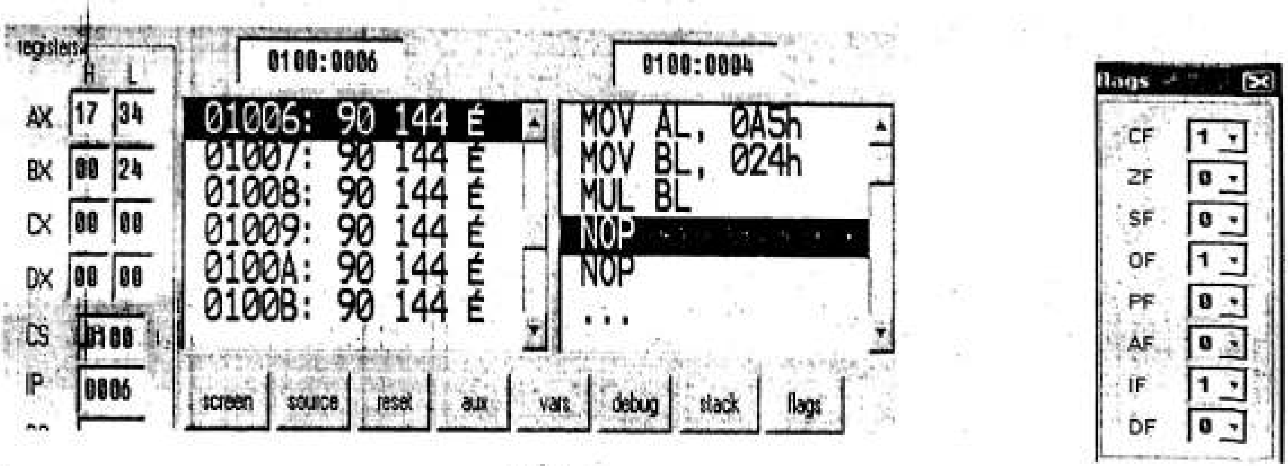
7

Signed Multiplication

```
0000   B0 A5   MOV AL,A5H     ;AL=A5H
0002   B3 24   MOV BL,24H     ;BL=24H
0004   F6 EB   IMUL BL        ;AX=ALxBL
```

product = F334H is in AX





A5h $\xrightarrow{\text{unsig}}$ 165
A5h $\xrightarrow{\text{sig}}$ -91

24h $\xrightarrow{\text{uns}}$ 36
24h $\xrightarrow{\text{sig}}$ 36

$$\begin{array}{r} 5B \\ \times\ 24 \\ \hline 16C \\ B6\ \ \\ \hline 0CCC \end{array}$$

F3 24

$$\begin{array}{r} A5 \\ \times\ 24 \\ \hline 294 \\ 14A\ \ \\ \hline 1734 \end{array}$$

165

165 × 31 = 5940
-91 × 36 = -3276

S

AAM : ASCII Adjust after Multiplication

This instruction, after execution, converts the product available In AL into unpacked BCD format.

AAM	No operands	ASCII Adjust after Multiplication. Corrects the result of multiplication of two BCD values. Algorithm: • AH = AL / 10 • AL = remainder C Z S O P A ? r r ? r ?

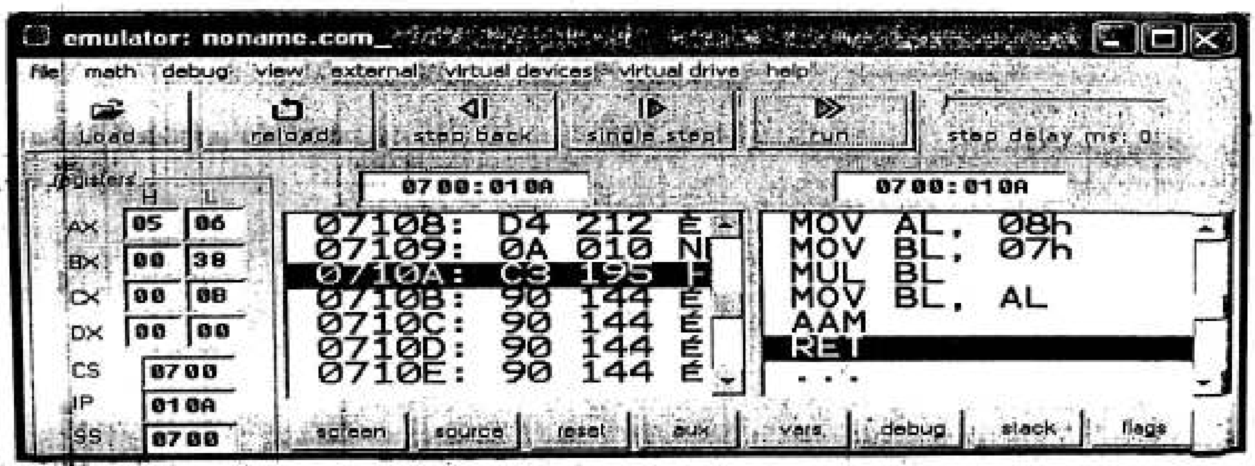
- Adjusts the result of the multiplication of two unpacked BCD values to create a pair of unpacked BCD values.
- The AX register is the implied source and destination.
- The AAM instruction is only useful when it follows an MUL instruction.

```
Eg.     MOV AL, 04   ;   AL = 04
        MOV BL ,09   ;   BL = 09
        MUL BL       ;   AX = AL*BL   ; AX=24H
        AAM          ;   AH = 03,  AL=06
```

BCD Multiplication

```
  Decimal          Binary(Hex)              ASCII
     8                 08                     08
    x7                x07                    x07
    56                 38               05    06
                                        AH    AL
```

```
0000   B0 08    MOV   AL,08H     ;AL = 08H
0002   B3 07    MOV   BL,07H     ;BL = 07H
0004   F6 E3    MUL   BL         ;AL = AL x BL
0006   D4 0A    AAM              ;ASCII adjust
```



❖ **Example:**

```
MOV AL,'7'        ; AH = 00H
AND AL, 0F        ; AL=07 unpacked BCD
MOV DL,'6'        ; DL=36H
AND DL,0FH        ; DL=06 unpacked BCD
MUL DL            ; AX=ALxDL=07x06
                      =002AH=42
AAM               ; AX=0402 (7x6=42
                    unpacked BCD)
OR AX, 3030H      ; AX=3432H result in
                    ASCII
```

❖ **Example:**

```
MOV BL,5          ; BL = 5H
MOV AL, 6         ; AL = 6H
MUL DL            ; AX=ALxDL=05x06
                      =001EH=30H
AAM               ; AX=0300
```


10

Arithmetic Instructions
DIV/IDIV: Unsigned/Signed Division

- **Division**
- It is an unsigned/signed division instruction.
- Occurs on 8- or 16-bit and 32-bit numbers depending on microprocessor.
- Signed (IDIV) or unsigned (DIV) integers.
- It divides word by byte or double word by word.
- There is no immediate division instruction available to any microprocessor.

- A division can result in two types of errors:
  - Attempt to divide by zero
  - Other is a divide overflow, which occurs when a small number divides into a large number. (ex: AX=3000/2 the result 1500 in AL cause and overflow).
- In either case, the microprocessor generates an interrupt if a divide error occurs.
- In most systems, a divide error interrupt displays an error message on the video screen.

- The following table shows the relationship between the dividend, divisor, quotient, and remainder.

Dividend	Divisor	Quotient	Remainder
AX	register or memory8	AL	AH
DX:AX	register or memory16	AX	DX
EDX:EAX	register or memory32	EAX	EDX

11

DIV Source

- Perform unsigned division operation
- If source operand is a byte,

  AL = AX / Source ; AH = Remainder of AX / Source
- If source operand is a word,

  AX=(DX AX)/Source ; DX=Remainder of (DX AX)/Source
- Source operands can not be an immediate data

Assembly Language Instruction	Operation
DIV rm8	AL = AX/rm8 (unsigned) AH = remainder
DIV rm16	AX = DX:AX/rm16 (unsigned) DX = remainder
IDIV rm8	AL = AX/rm8 (signed) AH = remainder
IDIV rm16	AX = DX:AX/rm16 (signed) DX = remainder

C	Z	S	O	P	A
?	?	?	?	?	?

?: flag value is undefined (maybe zero)

❖ **The following table shows the relationship between the dividend, divisor, quotient, and remainder.**

Dividend	Divisor	Quotient	Remainder
AX	register or memory8	AL	AH
DX:AX	register or memory16	AX	DX
EDX:EAX	register or memory32	EAX	EDX

ref

Assembly Language	*Operation*
DIV CL	AX is divided by CL; the unsigned quotient is in AL and the remainder is in AH
DIV BYTE PTR[BP]	AX is divided by the byte contents of the stack segment memor location addressed by BP; the unsigned quotient is in AL and the remainder is in AH
DIV CX	DX–AX is divided by CX; the unsigned quotient is in AX and the remainder is in DX

12

Dividend = BC2F
Divisor = EE
Quotient = CA
Remainder = 63

```
        CA
EE | BC2F
     B28
     ---
      9AF
      94C
      ---
       63
```

```
0000   B8 2F BC    MOV AX,0BC2FH     ;AX=BC2FH
0003   B3 EE       MOV BL,0EEH       ;BL=EEH
0006   F6 F3       DIV BL            ;AL=AX/BL
                   quotient  = AL = CAH
                   remainder = AH = 63H
```

- **The following instructions perform 8-bit unsigned division (83H/2), producing a quotient of 41H and a remainder of 1:**

```
MOV AX, 0083H        ; dividend
MOV BL, 02H          ; divisor
DIV BL               ; AL=41H, AH=01H
```

AX		BL		AL	AH
0083	/	02	→	41	01
				quotient	remainder

- **The following instructions perform 16-bit unsigned division (8003H/100H), producing a quotient of 80H and a remainder of 3. DX contains the high part of the dividend, so it must be cleared before the DIV instruction executes:**

```
MOV DX, 00           ; clear dividend, high
MOV AX, 8003H        ; dividend
MOV CX, 100H         ; divisor
DIV ~~BX~~ CX        ; AX=0080H, DX=0003H
```

DX	AX		CX		AX	DX
0000	8003	/	0100	→	0080	0003
					quotient	remainder

13

Byte / byte

- Numerator must be in AL and AH must be set to zero
- Denominator cannot be immediate but can be in memory or in a register.

word / word

- Numerator must be in AX and DX must be cleared
- Denominator can be in memory or in a register.
- After the division AX will have the quotient and DX will have the remainder

word / byte

- Numerator must be in AX
- Denominator can be in memory or in a register.
- After the division AL will have the quotient and AH will have the remainder

doubleword / word

- Numerator must be in AX and DX, least significant word in AX and most significant word in DX.
- Denominator can be in memory or in a register.
- After the division AX will have the quotient and DX will have the remainder

➢ **Example:**

```
                ; AX = 37D7H = 14, 295 decimal
                ; BH = 97H = 151 decimal
    DIV  BH     ;AX / BH
                ; AL = Quotient = 5EH = 94 decimal
                ; AH = Remainder = 65H = 101 decimal
```


14

- **IDIV Src**
- **The IDIV (signed divide) instruction performs signed integer division, using the same operands as DIV.**
- **Signed integers must be sign-extended before division takes place.**
- **Before executing 8-bit division, the dividend (AX) must be completely sign-extended. The remainder always has the same sign as the dividend.**
- **Fill high byte/word/doubleword with a copy of the low byte/word/doubleword's sign bit.**

-Perform signed division operation
- If source operand is a **byte**,

AL = AX / Source **; AH = Remainder of AX / Source**

- If source operand is a **word**,

AX=(DX AX)/Source **; DX=Remainder of (DX AX)/Source**

- Source operands can not be an immediate data

Assembly Language	*Operation*
DIV CX	DX–AX is divided by CX; the unsigned quotient is in AX and the remainder is in DX
IDIV SI	DX–AX is divided by SI; the signed quotient is in AX and the remainder is in DX
DIV NUMB	AX is divided by the contents of the data segment memory location NUMB; the unsigned quotient is in AX and the reminder is in DX

IDIV (signed number division) According to Intel manual IDIV means "integer division". **Note** that all arithmetic instructions of 8086 are for integer numbers. For real numbers (i.e. 5.32) 8087 coprocessor is used.

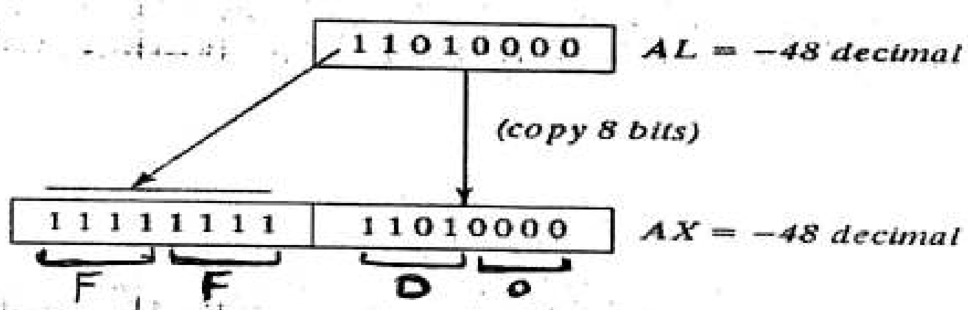
15

- **The following instructions divide-48 by 5. After IDIV executes, the quotient in AL is 9 and the remainder in AH is 3:**

48d

AL = D0H	**MOV AL,-48**	**; lower half of dividend in AL register**
Ax = FFD0	**CBW**	**; extend AL into AH**
BL = 5	**MOV BL, +5**	**; divisor**
AL = F7 = -9 AH = FD = -3	**IDIV BL**	**; AL=-9, AH=+3**

- **The following illustration shows how AL is sign-extended into AX by the CBW instruction:**



To understand why sign extension of the dividend is necessary, let's repeat the previous example without using sign extension. The following code initializes AH o zero so it has a known value, and then divides without using CBW to prepare the dividend:

MOV AH,00	**;upper half of dividend**
MOV AL, - 48	**;lower half of dividend in AL register**
MOV BL, +5	**;divisor**
IDIV BL	**; AL=41, AH=3**

AL = 29H = 41d
AH = 3

Before the division, AX = 00D0H (208 decimal). IDIV divides this by 5, producing a quotient of 41 decimal, and a remainder of 3. That is certainly not the correct answer.

-48d = D0H

16

- **16-bit division requires AX to be sign-extended into DX. The following instructions divide 5000 by 256. After IDIV executes, the quotient in AX is -19 and the remainder in DX is -136:**

```
MOV AX,-5000        ; lower half of dividend
                      in AX register
CWD                 ; extend AX into DX
MOV BX, +256        ; divisor
IDIV BX             ; quotient AX= -19,
                      remainder DX= -136
```

➢ **Example:**

```
IDIV  BP      ;divide a Signed double word in DX and
              ;AX by signed word in BP
IDIV  BYTE PTR[BX]     ; divide AX by a byte at
                       ;offset [BX] in DS
```

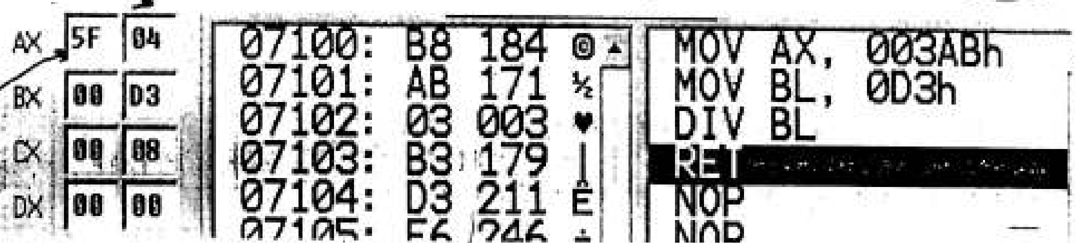
- **A signed word divided by a signed byte**

```
           ;AX = 00000011 10101011 = 03ABH=939 decimal
           ;BL = 11010011 = D3H = - 2DH = - 45 decimal
IDIV    BL ;Quotient AL= ECH = - 14H = -20 decimal
           ;Remainder AH = 27H = + 39 decimal
```

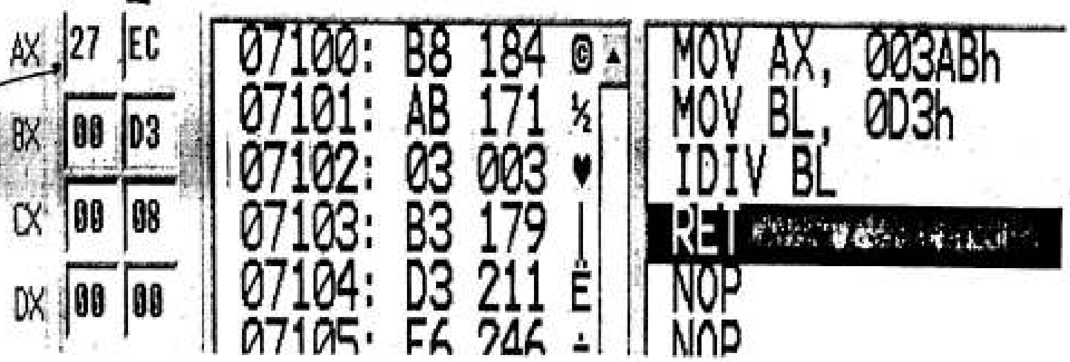
D3 $\xrightarrow{un}$ 211, D3 $\xrightarrow{sn}$ -45

03AB $\xrightarrow{s}$ +939 d, 03AB $\xrightarrow{u}$ +939 d

4
211) 939
844
95



-20
45) 939
90
039



20 → 00010100 $\xrightarrow[\text{Com}]{2,s}$ 11101100

or

14 $\xrightarrow[\text{Com}]{16,s}$ EC

39d = 00100111

17

AAD: ASCII Adjust before Division

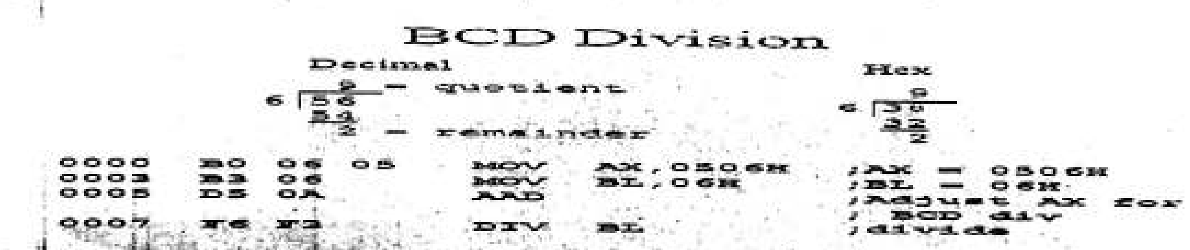
- This instruction converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. In the instruction sequence, this instruction appears Before DIV instruction.
- The AAD instruction requires the AX register contain a two- digit unpacked BCD number (not ASCII) before executing.
- Before dividing the unpacked BCD by another unpacked BCD, AAD is used to convert it to HEX. By doing that the quotient and reminder are both in unpacked BCD.

Eg. AX 05 08

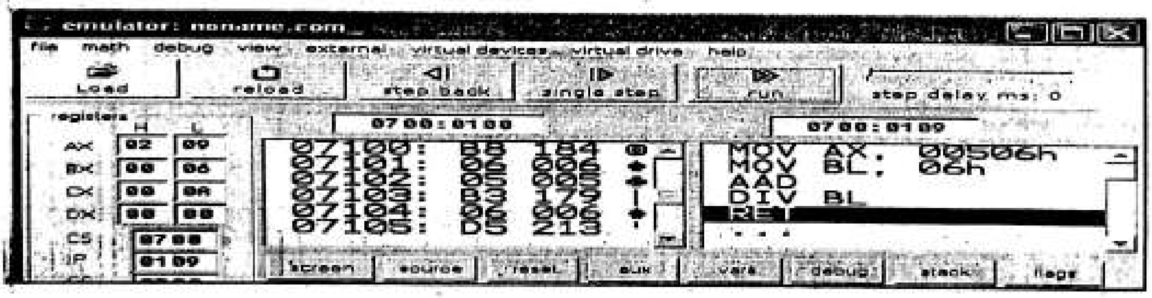
AAD result in AX 00 3A $58_D = 3A$ H in AL

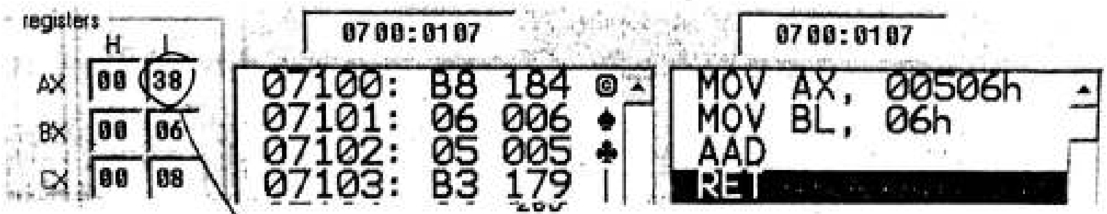
The result of AAD execution will give the hexadecimal number 3A in AL and 00in AH. Where 3A is the hexadecimal Equivalent of 58 (decimal).

AAD	No operands	ASCII Adjust before Division. Prepares two BCD values for division. Algorithm: • AL = (AH * 10) + AL • AH = 0



9
6)56
54
02





38 H = 56 D

Sign extending bytes to words

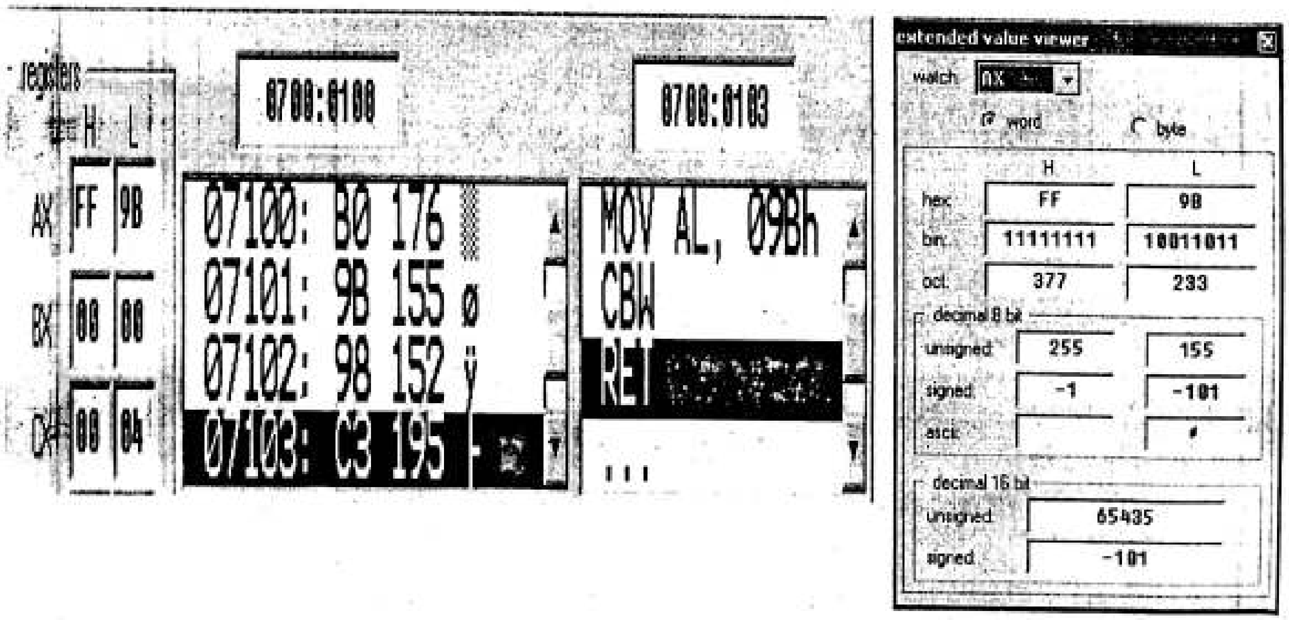
```
 5  =  00000101
-5  =  11111011  =  FBH
16 bits
-5  =  1111111111111011  =  FFFBH
```

To add an 8-bit signed number to a 16-bit signed number the 8-bit number must be sign extended:
If bit 7 is 1, make bits 8 - 15 one.
If bit 7 is 0, make bits 8 - 15 zero.

CBW : Convert byte to word

-Extends a signed 8-bit number in AL to a signed 16-bit data and stores it into AX
- It does not modify flags

CBW	No operands	Convert byte into word.
		Algorithm:
		if high bit of AL = 1 then:
		• AH = 255 (0FFh)
		else
		• AH = 0
		Example:
		`MOV AX, 0    ; AH = 0, AL = 0`
		`MOV AL, -5   ; AX = 000FBh (251)`
		`CBW          ; AX = 0FFFBh (-5)`
		`RET`
		C Z S O P A
		unchanged



19

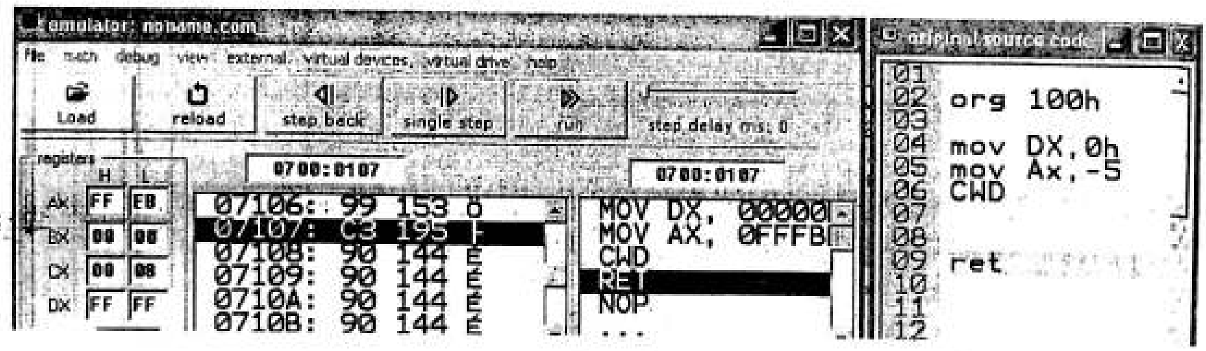
CWD : Convert Word to Double word.

- Extends a signed 16-bit number in AX to a signed 32-bit data and stores it into DX and AX. DX contains the most significant word
- It does not modify flags

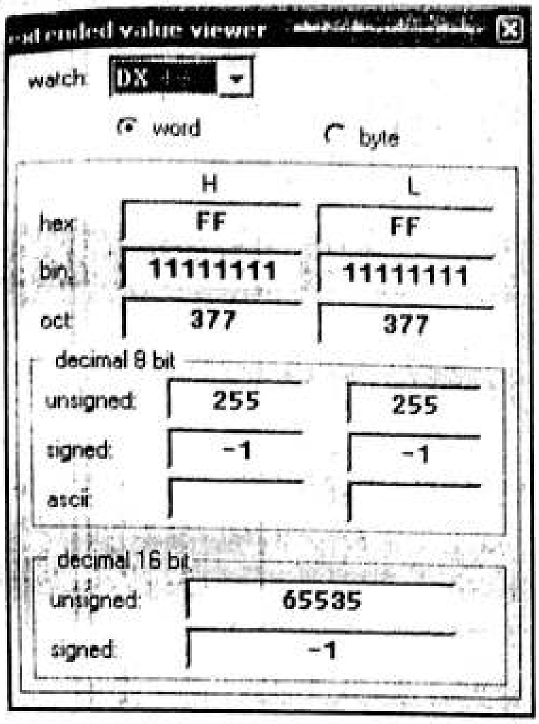
CWD	No operands	Convert Word to Double word.
		Algorithm:
		if high bit of AX = 1 then:
		• DX = 65535 (0FFFFh)
		else
		• DX = 0
		Example:
		MOV DX, 0 ; DX = 0
		MOV AX, 0 ; AX = 0
		MOV AX, -5 ; DX AX = 0000h:0FFFBh
		CWD ; DX AX = 0FFFFh:0FFFBh
		RET
		C Z S O P A
		unchanged

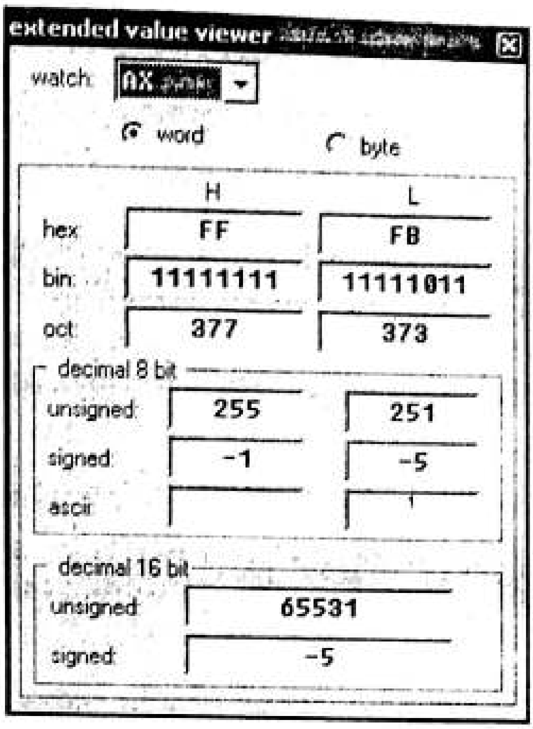
Example:

```
            ;DX = 00000000 00000000
            ;AX = 11110000 11000111 = - 3897 decimal
CWD         ;Convert signed word in AX to signed double
            ;word in DX:AX
            ;Result  DX = 11111111 11111111
            ;AX = 11110000 11000111 = -3897 decimal .
```



20

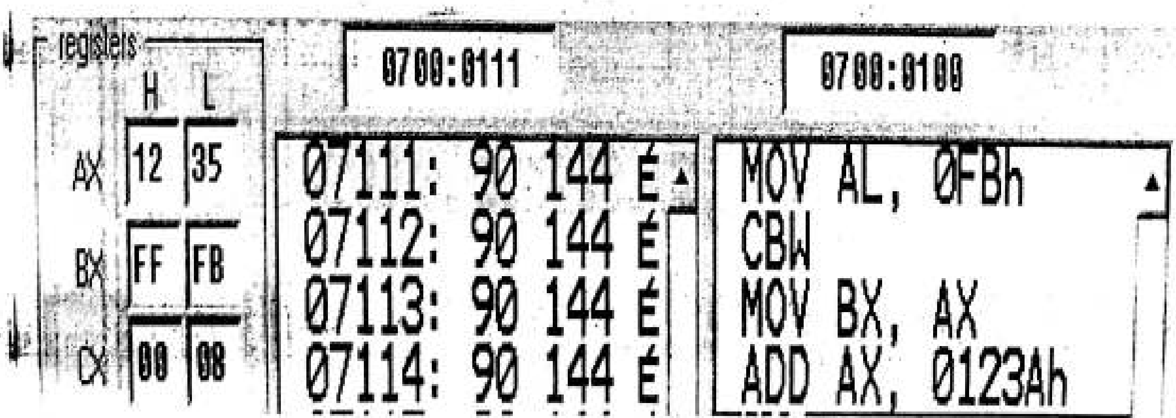




Add 8-bits to 16-bits (signed)

FBH +123AH	Sign extend	FFFBH +123AH 1235H

```
0000  B0 FB      MOV AL,FBH     ;AL = -5
0002  98         CBW            ;AX = -5
0003  05 3A 12   ADD AX,123AH   AX = sum
```



Mov AL, -5
ADD AX, 1234h
X [13 | 2F]

  00FB
+ 1234
  132F

Mov AL, -5
cbw
ADD AX, 1234h

  FFFB
+ 1234
  122F

1. Write an ALP (assembly language programming) for addition of two 8-bit data BB H and 11 H.

```
MOV AL, BB H      : 8-bit data BB H into AL
MOV CL, 11 H      : 8-bit data 11 H into CL
ADD AL, CL        : Contents of AL and CL added
HLT               : Stop.
```

Comment : Result in AL = CC H.

2. Write an ALP for addition of two 16-bit data BB11 H and 1122 H.

```
MOV AX, BB11 H    : 16-bit data BB11 H into AX
MOV CX, 1122 H    : 16-bit data 1122 H into CX
ADD AX, CX        : Contents of AX and CX added
HLT               : Stop
```

Comment : Result in AX = CC33 H.

3. Write an ALP for addition of two 8-bit data BB H and 11 H. The first data has an offset address of 0304 H and displacement 07.

```
MOV BX, 0304 H    : Offset address put in BX
MOV AL, 11 H      : 8-bit data 11H into AL
ADD AL, [BX + 07] : 8-bit data from offset + displacement added with AL
HLT               : Stop.
```

Comment : Result in AL = CC H.

4. Write an ALP that subtracts 1234 H existing in DX from the word beginning at memory location MEMWDS.

```
MOV DX, 1234 H    : 16-bit data 1234 H put into DX
SUB MEMWDS, DX    : Subtract data word 1234 H existing in DX from the data word
                    pointed to by MEMWDS.
HLT               : Stop.
```

Comment : If MEMWDS points to 3000 H then,
[3001 H : 3000 H] ← [3001 H : 3000 H] – 1234 H

5. Write an ALP which multiplies two 8-bit data 21 H and 17 H.

```
MOV AL, 21 H      : 8-bit multiplicand 21 H put into AL
MOV CL, 17 H      : 8-bit multiplier 17 H put into CL
MUL CL            : Contents of CL and AL are multiplied and the result
                    stored in AX
HLT : Stop.
```

Comment : Result in AX = 02F7 H.

6. Write an ALP for dividing 1234 H by 34 H.

```
MOV AX, 1234 H    : 16-bit dividend in 1234 H
MOV CL, 34 H      : 8-bit divisor in 34 H
DIV CL            : Content of AX divided by content of CL
HLT               : Stop.
```

Comment: Result in AX with Quotient in AL = 59 H and Remainder in AH = 20 H.

22

7. Write an ALP for ASCII addition of two numbers 2 H and 5 H.

```
MOV AL, 32 H      : ASCII code 32 H for number 2 H is moved into AL
MOV BL, 35 H      : ASCII code 35 H for number 5 H is moved into BL
AAA               : ASCII adjust for addition
HLT               : Stop.
```

Result : (AL) = 07 H.

8- Write an ALP to evaluate X (Y + Z), where X = 10 H, Y = 20 H and Z = 30 H.

```
MOV AL, 20 H      : 20 H put in AL
MOV CL, 30 H      : 30 H put in CL
ADD AL, CL        : AL and CL are added up and result in AL
MOV CL, AL        : AL transferred in CL
MOV AL, 10 H      : 10 H put in AL
MUL CL            : AL and CL are multiplied and result in AL
MOV SI, 4000 H    : Source address in SI
MOV SI, AL        : AL put in SI
HLT : Stop.
```

9- Write an ALP to evaluate X^3 +10 , X, is present in the data segment of memory at offset 2000h and store the result in 3000h.

```
MOV  AL  , [2000h]
MOV  AH  , 0h
MOV  BL  , AL
MUL  AL
MUL  BL
ADD  AX  ,000Ah
MOV [3000h]  ,AX
```

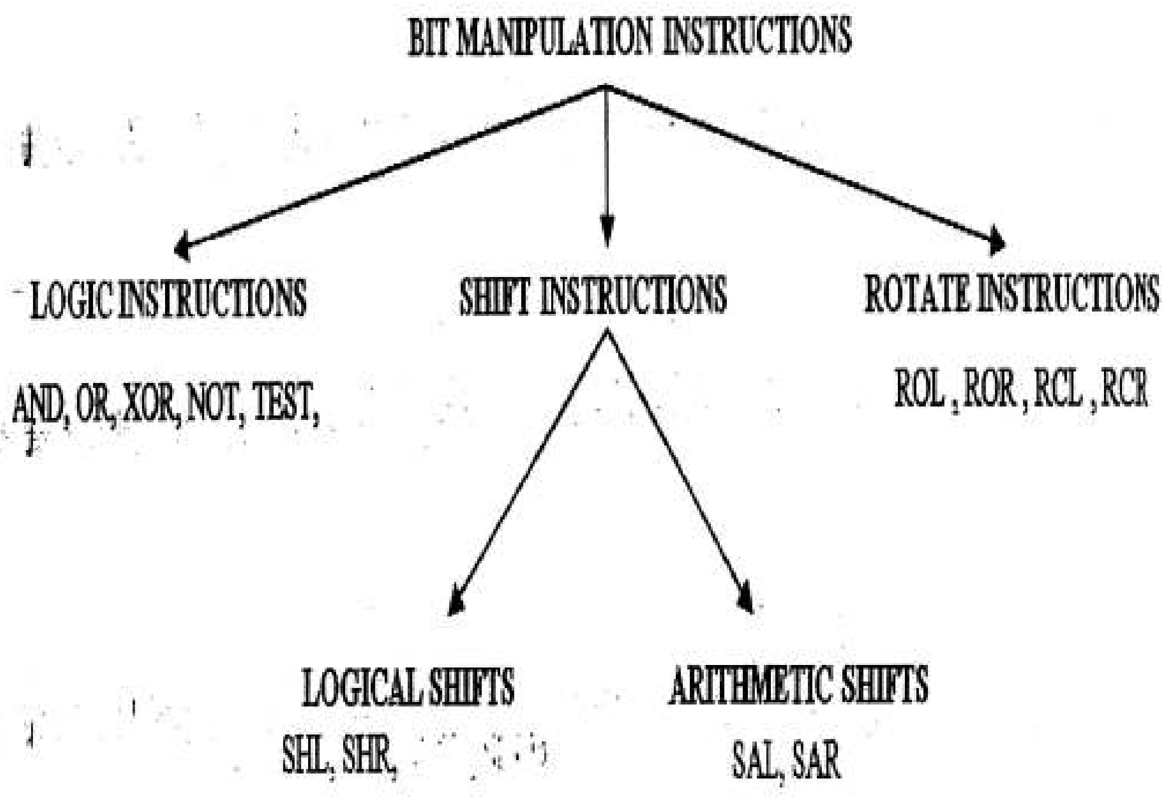
10- Write an ALP that transfers a block of 100 bytes of data. The source and destination memory blocks start at 3000 H and 4000 H memory locations respectively. The data segment register value is 1000h.

```
    MOV AX, 1000h   : Move initial address of DS register into AX.
    MOV DS, AX      : DS loaded with AX
    MOV SI, 3000 H  : Source address put into SI.
    MOV DI, 4000 H  : Destination address put into DI.
    MOV CX, 64 H    : Count value for number of bytes put into CX register
xx: MOV AH, [SI]    : Source byte moved into AH
    MOV [DI], AH    : AH byte moved into destination address
    INC SI          : Increment source address
    INC DI          : Increment destination address
    DEC CX          : Decrement CX count
    JNZ xx          : Jump to 200D H until CX = 0
    HLT : Stop.
```


23

BIT MANIPULATION INSTRUCTIONS

LOGICAL INSTRUCTIONS	SHIFT INSTRUCTIONS	ROTATE INSTRUCTIONS
NOT AND OR XOR TEST	SHL / SAL SHR SAR	ROL ROR RCL RCR



- The logic instructions include
  - AND
  - OR
  - XOR (Exclusive-OR)
  - NOT

Mnemonic	Meaning	Format	Operation	Flags affected
AND	Logical AND	AND D, S	(S) · (D) → (D)	OF, SF, ZF, PF, CF AF undefined
OR	Logical Inclusive-OR	OR D, S	(S) + (D) → (D)	OF, SF, ZF, PF, CF AF undefined
XOR	Logical exclusive-OR	XOR D, S	(S) ⊕ (D) → (D)	OF, SF, ZF, PF, CF AF undefined
NOT	Logical NOT	NOT D	(NOT D) → (D)	None

- Logic instructions : AND, OR, XOR, NOT

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

Allowed operands for AND, OR, and XOR instructions

Destination
Register
Memory

Allowed operands for NOT instruction

Logical Operations.

y	x	x AND y	x OR y	x XOR y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

25

Logical Instructions

→ **Include AND, OR, Exclusive- OR, and NOT.**
→ **These instructions are used at the bit level.**
→**These instructions can be used for: -**
- **Testing a zero bit**
- **Set or reset a bit**
- **Shift bits across registers**

→ **Logic operations provide binary bit control in low- level software. Allow bits to be set, cleared, or complemented.**

AND Destination, Source

→ The AND instruction performs a boolean (**bitwise**) AND operation between each pair of matching bits in two operands and places the result in the destination operand.
→ The following operand combinations are permitted:

```
AND reg, reg
AND reg, mem
AND reg, imm
AND mem, reg
AND mem, imm
```

→ In8086, the AND instruction often executes in about a microsecond. With newer versions, the execution speed is greatly increased.

→ AND clears bits of a binary number called <u>masking</u>.
→ AND uses any mode except memory- to- memory and segment register addressing.
→ An ASCII number can be converted to BCD by using AND to mask off the leftmost four binary bit positions.

26

	x x x x x x x x	Unknown number
MOV BX, 3135H **AND BX, 0F0FH**	• 0 0 0 0 1 1 1 1	Mask
	0 0 0 0 x x x x	Result

→ CF and OF become **zero** after the operation.

→ PF, SF and ZF are **updated**.

→ The AND instruction lets you clear 1 or more bits in an operand without affecting other bits. The technique is called bit masking.

Assembly Language	*Operation*
AND AL,BL	AL = AL AND BL
AND CX,DX	CX = CX AND DX
AND ECX,EDI	ECX = ECX AND EDI
AND CL,33H	CL = CL AND 33H
AND DI,4FFFH	DI = DI AND 4FFFH
AND ESI,34H	ESI = ESI AND 00000034H
AND AX,[DI]	AX is ANDed with the word contents of the data segment memory location addressed by DI
AND ARRAY[SI],AL	The byte contents of the data segment memory location addressed by the sum of ARRAY plus SI is ANDed with AL; the result moves to memory
AND [EAX],CL	CL is ANDed with the byte contents of the data segment memory location addressed by EAX; the result moves to memory

OR Destination, Source

- Performs logic OR operation for each bit of the destination and source; stores the result into destination
- Destination and source can not be both memory locations at the same time
- It modifies flags: CF OF PF SF ZF.**(CF=OF=0).**

x x x x x x x x	Unknown number
+ 0 0 0 0 1 1 1 1	Mask
x x x x 1 1 1 1	Result

- **Suppose AL is initially equal to 11100011 binary and then we OR it with 00000100, AL equals 11100111:**

MOV AL, 11100111b

OR AL, 00000100b ; result in AL=11100111b

XOR Destination, Source

- Performs logic XOR operation for each bit of the destination and source; stores the result into destination
- Destination and source can not be both memory locations at the same time
- It modifies flags: CF OF PF SF ZF.**(CF=OF=0)**

x x x x x x x x	Unknown number
$\oplus$ 0 0 0 0 1 1 1 1	Mask
x x x x $\bar{x}\,\bar{x}\,\bar{x}\,\bar{x}$	Result

28

❖ **Example:**

MOV AL, 54H ; **AL=01010100 b**

XOR AL, 87H ; **result in AL=00101100 b**

54H	0101 0100
78H	0111 1000
2CH	0010 1100

❖ **Example: Clearing the contents of register.**

MOV AL, 54H ; **AL=01010100 b**

XOR AL, AL ; **result in AL=00000000 b**

❖ **Flags: SF = CF = OF = 0; ZF = PF = 1**

54H	0101 0100
54H	0101 0100
00H	0000 0000

❖ **Example: Bit toggle.**

MOV AL, 54H ; **AL=01010100 b**

XOR AL, 04H ; **Toggle bit No. 2**

54H	0101 0100
04H	0000 0010
56H	0101 0110

- **NOT Src:**
- **It complements each bit of Src to produce one's complement of the specified operand.**
- **The operand can be a register or memory location.**
- **NOT can use any addressing mode except segment register addressing.**
- **The NOT function is considered logical, NEG function is considered an arithmetic operation.**
- **None of flags are affected by NOT instruction.**

- **The NOT instruction toggles (inverts) all bits in an operand.**
- **The following operand combinations are permitted:**

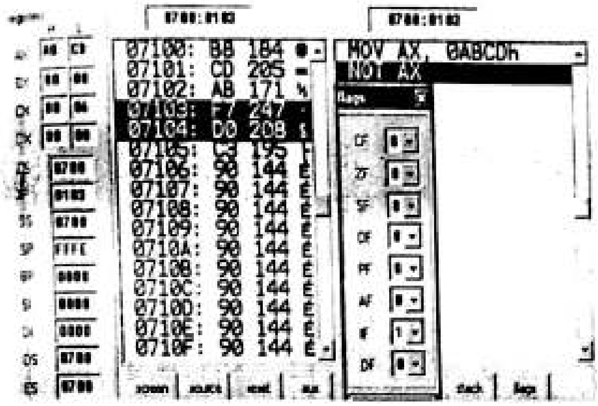
```
NOT  reg
NOT  mem
```

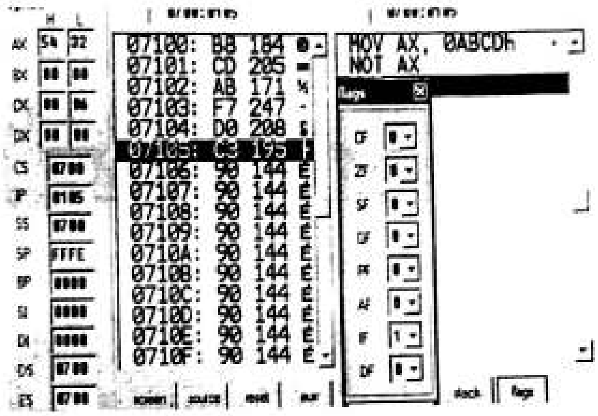
- **Example: The one's complement of F0h is 0Fh:**

```
MOV  AL, F0H        ; AL=11110000 b
NOT  AL             ; AL=00001111 b
```

C	Z	S	O	P	A
unchanged					

- It does not modify flags





Setting, clearing and inverting selected bits in the Destination operand

* **Mask :** clear a bit to zero . To **clear** a bit, we AND it with zero. AND with 1 **: unchanged**
* **Set**: Setting a bit to 1: OR with logic 1.
* **Toggle** : XOR to reverse the login level

If X represents a bit (0 or 1) then:

X AND 0 = 0	X OR 0 = X	X XOR 0 = X
X AND 1 = X	X OR 1 = 1	X XOR 1 = $\overline{X}$

❖ **Example:**

```
MOV AL, 55H     ; AL=01010101b
AND AL, 1FH     ; AL=15H=00010101b,
                  clear bit 7
OR AL, C0H      ; AL=D5H=11010101b,
                  set bits 1, 3, 5, 7, 8
XOR AL, 0FH     ; AL=DAH=11011010b,
                  invert bits 1, 2, 3, 4
NOT AL          ; AL=25H=00100101b,
                  toggles (invert) all bits
```


31

Ex: MOV DH,54H
XOR DH,78H

Solution:		
54H	01010100	
78H	01111000	
2C	00101100	SF=0, ZF=0, PF=0, CF=OF=0

Ex: Assume CH=35H

Solution: XOR CH,35H

35H	00110101	
35H	00110101	
00	00000000	SF=0, ZF=1, PF=1, CF=OF=0

Ex: Change bit2 (D3) in BL to the opposite value and all other bits would remain unchanged.

XOR BL,04H ; XOR BL with 000 0100

Solution: This will cause bit 2 (D3) of BL to change to the opposite value; all other bits would remain unchanged.

Example

```
OR    CX,FF00h    ;OR CX with immediate FF00h
                  ;result in CX = 11111111 10100101
                  ;Upper byte are all 1's lower bytes
                  ;are unchanged.
```

Ex

Clear a general-purpose register using a four different instructions

```
MOV  AX , 0
SUB  BX , BX
AND  CX , 0
XOR  DX , DX
```


32

EXAMPLE

Describe the results of executing the following instructions'

```
MOV  AL, 01010101B
AND  AL, 00011111B
OR   AL, 11000000B
XOR  AL, 00001111B
NOT  AL
```

Solution:

$(AL)=01010101_2 \cdot 00011111_2= 00010101_2=15_{16}$

Executing the OR instruction, we get

$(AL)= 00010101_2 +11000000_2= 11010101_2=D5_{16}$

Executing the XOR instruction, we get

$(AL)= 11010101_2 \oplus 00001111_2= 11011010_2=DA_{16}$

Executing the NOT instruction, we get

$(AL)= (NOT)11011010_2 = 00100101_2=25_{16}$

EXAMPLE

Masking and setting bits in a register.

Solution:

Mask off the upper 12 bits of the word of data in AX

AND AX, $000F_{16}$

Setting B_4 of the byte at the offset address CONTROL_FLAGS

```
MOV AL, [CONTROL_FLAGS]
OR AL, 10H
MOV [CONTROL_FLAGS], AL
```

Executing the above instructions, we get

$(AL)=XXXXXXXX_2 +00010000_2= XXX1XXXX_2$

23

Changing a letter to its opposite case

- Suppose **CL** contains a lowercase alphabetic letter. To change that letter to uppercase, we subtract 20H from **CL**:

```
SUB  CL , 20H
```

- Suppose BL contains an uppercase alphabetic letter. To change that letter to lowercase, we add 20H to BL:

```
ADD  BL , 20H
```

- For any alphabetic letter, bit 5 of its ASCII code is 1; but for the corresponding uppercase letter bit 5 is 0. The remaining bits are similar:

Letter		ASCII code	Letter		ASCII code
'a'	61 H	0110 0001B	'A'	41 H	0100 0001B
'b'	62 H	0110 0010B	'B'	42 H	0100 0010B
'c'		0110 0011B	'C'		0100 0011B
.			.		
.			.		
.			.		
'y'		0111 1001B	'Y'		0101 1001B
'z'		0111 1010B	'Z'		0101 1010B

Thus a lowercase alphabetic letter can also be converted to uppercase by clearing bit 5 of its ASCII code. This can be done by using an AND instruction with the mask 11011111B or 0DFh. Example:

```
MOV  DL , 'j'

AND  DL , 11011111B
```

An uppercase alphabetic letter can also be converted to lowercase by setting bit 5 of its ASCII code. This can be done by using an OR instruction with the mask 00100000B or 20H. Example:

```
MOV  AL  ,  'M'

OR  AL , 00100000B
```

To convert a lowercase or uppercase letter to its opposite case we need only invert bit 5 of its ASCII code. This can be done by using an XOR instruction with the mask 00100000B.

TEST : Logical Compare Instruction

- The **TEST instruction** performs the AND operation. The difference is that The AND instruction changes the destination operand, while the TEST instruction dose not. A TEST affects only condition flags.

- The TEST instruction functions in the same manner as a CMP instruction. The difference is that the TEST instruction normally tests a single bit (or occasionally multiple bits), while the CMP instruction tests the entire byte or word.

C	Z	S	O	P
0	✓	✓	0	✓

✓ : update

Assembly Language	*Operation*
TEST DL,DH	DL is ANDed with DH
TEST CX,BX	CX is ANDed with BX
TEST EDX,ECX	EDX is ANDed with ECX
TEST AH,4	AH is ANDed with 4

- **The zero flag (Z)** is a **logic 1** (***indicating a zero result***) if the bit under test is zero, and **Z=0** (indicating a non- zero result) if the bit under test is non zero.

```
0000  A8 01        TEST  AL,1      ;test right bit
0002  75 1C        JNZ   RIGHT     ;if set
0004  A8 80        TEST  AL,128    ;test left bit
0006  75 38        JNZ   LEFT      ;if set
```

Ex

Mov AL, 00000101b

Test AL,1 ZF=0

Test AL,10b ZF=1

Shift and Rotate Instructions

→ Shift and Rotate instructions manipulate binary numbers at the binary bit level.

Shift instructions are:

- Shift Logical Left (SHL).
- Shift Logical Right (SHR).
- Shift Arithmetic Left (SAL).
- Shift Arithmetic Right (SAR).

Format: SAR/SHR/SAL/SHL D , COUNT.

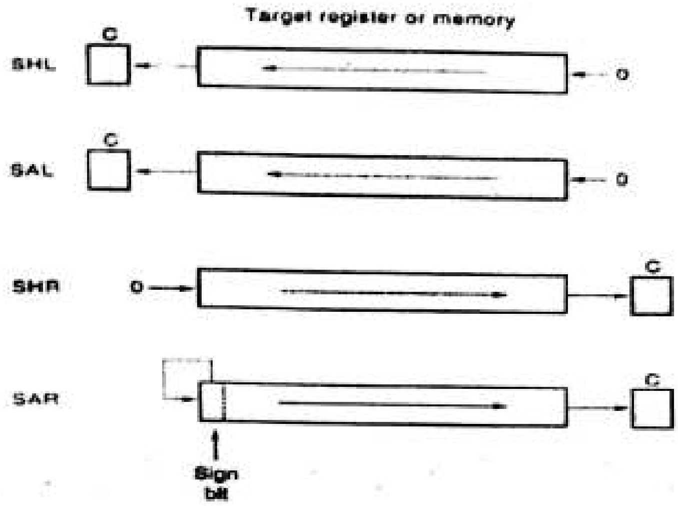
- **Shift: position or move numbers to the left or right within a register or memory location.**
- **The microprocessor's instruction set contains four different shift instructions:**
  - **Two Logical shift**
  - **Two Arithmetic shift**
- **Logical shift function with unsigned numbers.**
- **Arithmetic shift function with signed numbers.**

1- **Logical shifts move 0 in the rightmost bit for a logical left shift.**

2- **The arithmetic shift left is identical to the logical shift left.**

3- **0 to the leftmost bit position for a logical right shift.**

4- **The arithmetic right shift copies the sign-bit through the number.**



- Shifting means to move bits right and left inside an operand.
- All four shifting instruction affecting the Overflow and Carry flags.

Mnemonic	Meaning
SHL	Shift left
SHR	Shift right
SAL	Shift arithmetic left
SAR	Shift arithmetic right

- Logical shifts multiply or divide unsigned data; arithmetic shifts multiply or divide signed data.
  - A shift left always multiplies by 2 for each bit position shifted.
  - A shift right always divides by 2 for each position.
  - Shifting a two places, multiplies or divides by 4.
- Segment shift not allowed.

SHL(SAL) ***Destination, Count***

- Left shift destination bits; the number of bits shifted is given by operand Count.
- SAL and SHL are two mnemonics for the same instruction.
- During the shift operation, the MSB of the destination is shifted into CF and zero is shifted into the LSB of the destination
- Operand Count can be either an immediate data or register CL
- Destination can be a register or a memory location
- It modifies flags: CF OF PF SF ZF
- OF=0 if the first operand keeps original sign.

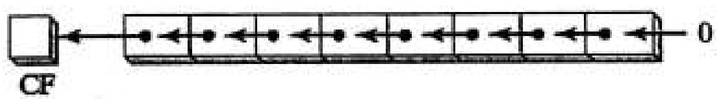
❖ **The following lists the types of operands permitted by this instruction:**

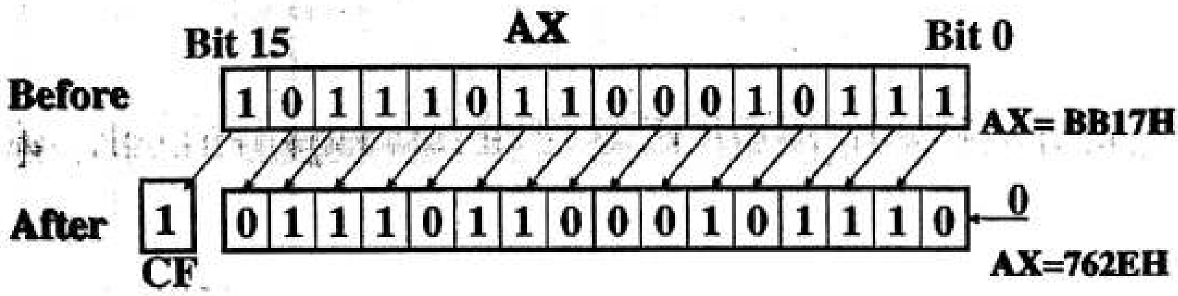
SHL reg, imm8
SHL mem, imm8
SHL reg, CL
SHL mem, CL

❖ **The first operand in SHL is the destination and the second is the shift count.**

❖ **Example:**

MOV AX, BB17H ; AX=1011101100010111b
SHL AX, 1 ; AX=0111011000101110b, CF=1





Ex:
```
MOV  DH,6
MOV  CL,4            ;set number of times to shift
SHL  DH,CL
```

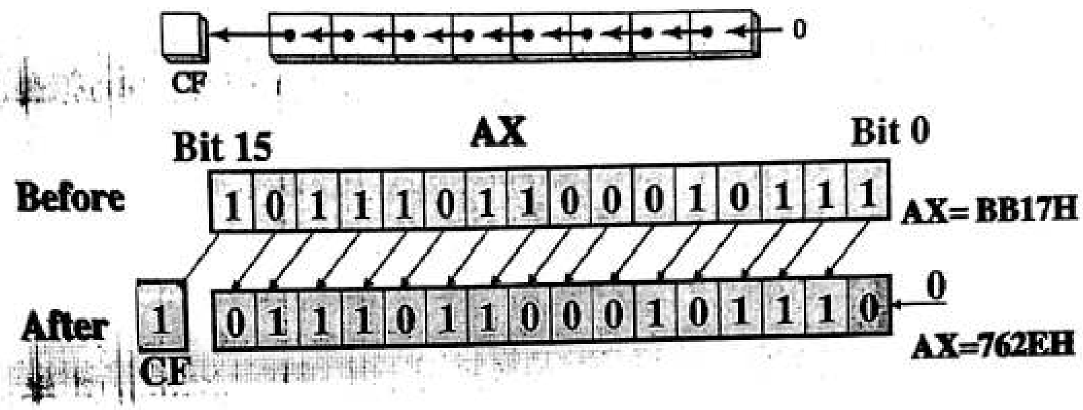
Solution:	00000110	
CF=0	00001100	(shifted left once)
CF=0	00011000	
CF=0	00110000	
CF=0	01100000	(shifted left 4 times)

After the 4 shifts DH=60H and CF=0.

❖ **Example:**

MOV AX, BB17H ; AX=1011101100010111b

SAL AX, 1 ; AX=0111011000101110b, CF=1



❖ **Example: In the following instructions, If binary 01000000 (decimal 64, 40H) is shifted right by 3 bits, the result is the same as dividing 64 by 2^3:**

Shift Right

MOV DL, 40H ; DL=01000000b

SHR DL, 3 ; DL=00001000b, CF=0

Before: DL=01000000b
After: DL=00001000b, CF=0

Ex

Mov AL, 40H

SHL AL, 1

~~SHL AL, 1~~

AL=80H OF=1 SF=1 CF=0 ZF=0

Befor

AL=40H [0 1 0 0 0 0 0 0]

+

After

AL=80H [1 0 0 0 0 0 0 0]

−

OF=1

Note

SHL = SAL

❖ Example: In the following instructions, BL is shifted once to the left. The highest bit is copied into the Carry flag and the lowest bit position is assigned zero:

```
MOV  BL, F0H        ; BL=11110000b
SAL  BL, 1          ; BL=11100000b, CF=1
```

Before: BL=11110000b (-16 decimal, F0H)
After: BL=11100000b (-32 decimal, E0H) , CF=1

❖ Example: In the following instructions, AL is shifted arithmetic twice to the left, bit 7 does not end up in the Carry flag because it is replaced by bit 6 . After SAL, AL=80H (-128d)

```
MOV  AL, E0H        ; AL=11100000b (-32d)
SAL  AL, 2          ; AL=10000000b, CF=1
```

(-128 d)

$-32 * 2^2 = -32 * 4 = -128$

٤٥

Shifting left 1 bit multiplies a number by 2

```
mov dl,5
shl dl,1
```

Before:	00000101	=5
After:	00001010	=10

Shifting left *n* bits multiplies the operand by 2^n

For example, $5 * 2^2 = 20$

```
mov dl,5
shl dl,2          ; DL = 20
```

```
AX | 00 | 40     07100: B0 176      MOV AL, 010h
BX | 00 | 04     07101: 10 016 ►    MOV BL, AL
CX | 00 | 00     07102: 8A 138 è    SHL AL, 1
DX | 00 | 00     07103: D8 216 Ï    SHL AL, 1
                 07104: D0 208 ß    SHR BL, 1
                 07105: E0 224 ó    SHR BL, 1
```

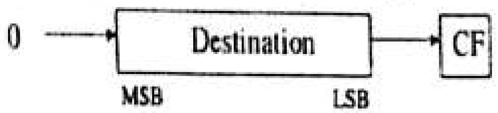
EXAMPLE

```
                    ;Multiply AX by 10 (1010)
                    ;
0000  D1 E0              SHL   AX,1         ;AX times 2
0002  8B D8              MOV   BX,AX
0004  C1 E0 02           SHL   AX,2         ;AX times 8
0007  03 C3              ADD   AX,BX        ;10 times AX
                    ;
                    ;Multiply AX by 18 (10010)
                    ;
0009  D1 E0              SHL   AX,1         ;AX times 2
000B  8B D8              MOV   BX,AX
000D  C1 E0 03           SHL   AX,3         ;AX times 16
0010  03 C3              ADD   AX,BX        ;18 times AX
                    ;
                    ;Multiply AX by 5 (101)
                    ;
0012  8B D8              MOV   BX,AX
0014  D1 E0              SHL   AX,1         ;AX times 2
0016  D1 E0              SHL   AX,1         ;AX times 4
0018  03 C3              ADD   AX,BX        ;5 times AX
```


41

SHR Destination, Count

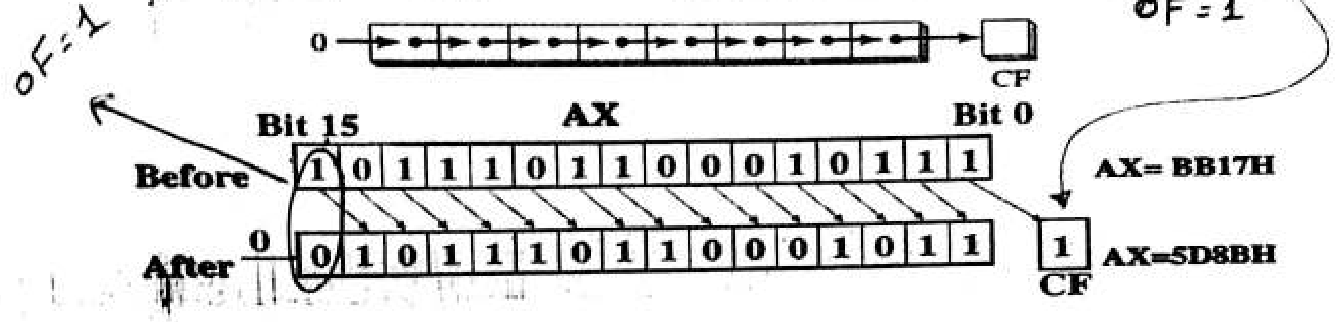
- Right shift destination bits; the number of bits shifted is given by operand Count
- During the shift operation, the LSB of the destination is shifted into CF and zero is shifted into the MSB of the destination
- Operand Count can be either an immediate data or register CL
- Destination can be a register or a memory location
- It modifies flags: CF OF PF SF ZF
- OF = 0 if first operand keeps orginal sign.



❖ **Example:**

MOV AX, BB17H ; AX=1011101100010111b
SHR AX, 1 ; AX=0101110110001011b, CF=1

OF = 1



❖ **Example: In the following instructions, shifting the integer 32 decimal (20H) right by 2 bit yields the division of $32/2^2 = 8$**

MOV DL, 20H ; DL=00100000b
SHR DL, 2 ; DL=00001000b, CF=0

```
Ex:        MOV   AL,9AH
           MOV   CL,3                              ;set number of times to shift
           SHR   AL,CL

Solution:        9AH   10011010
                       01001101        CF=0  (shifted once)
                       00100110        CF=1  (shifted twice)
                       00010011        CF=0  (shifted three times)
```

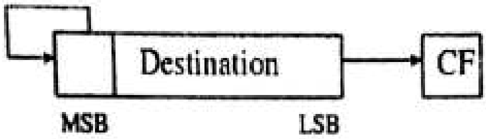
After three times of shifting AL=13H and CF=0

42

Scanned by CamScanner

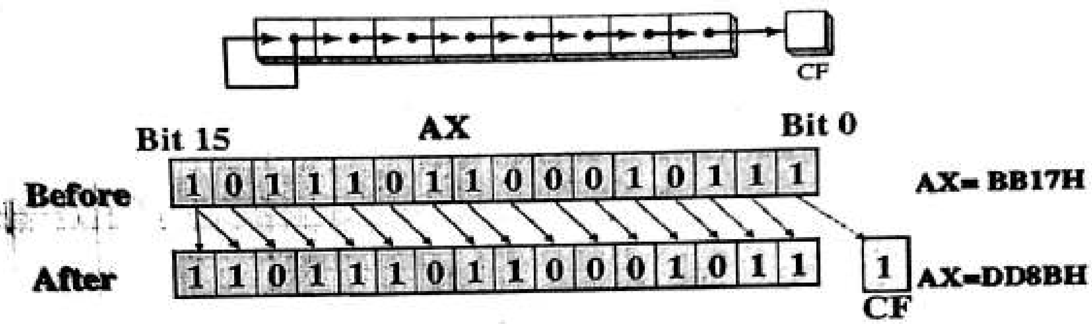
SAR Destination, Count

- Right shift destination bits; the number of bits shifted is given by operand Count
- The LSB of the destination is shifted into CF and the MSB of the destination remains the same
- Operand Count can be either an immediate data or register CL
- Destination can be a register or a memory location
- It modifies flags: CF PF SF ZF



- **Example:**

```
MOV  AX, BB17H     ; AX=1011101100010111b
SAR  AX, 1         ; AX=1101110110001011b, CF=1
```



- **Example: In the following instructions, AL is shifted once to the right. The lowest bit is copied into the Carry flag and the highest bit position is filled with the value of the original MSB :**

```
MOV  AL, F0H     ; AL=11110000b, (-16d)
SAR  AL, 1       ; AL=11111000b, (-8d) CF=0
```

Before: AL=11110000b (-16 decimal, F0H)
After: AL=11111000b (-8 decimal, F8H) , CF=0

43

- Example: In the following instructions, AL is shifted arithmetic triple to the right. -128 is divided by 2^3. After SAR, AL=F0H (-16d).

```
MOV  AL, 80H    ; AL=10000000b (-128d)
SAR  AL, 3      ; AL=11110000b (-16d), CF=0
```

```
                 ;CF = 0, BX = 11100101  11010011
SAL      BX, 1   ;Shift BX register contents by 1 bit
                 ;position towards left
                 ;CF = 1, BX = 11001011  1010011

SAR      AL, 1   ;Shift signed byte in AL towards right
                 ;( divide by 2 )
                 ;AL = 00001110 = + 14 decimal, CF = 1
         ( 2 )
                 ;BH = 11110011 = - 13 decimal, CF = 1
SAR      BH, 1   ;Shifted signed byte in BH  to right
                 ;BH = 11111001 = - 7 decimal, CF = 1

                 ;AL = 01010001
TEST     Al, 80H ;AND immediate 80H with AL to
                 ;test f MSB of AL is 1 or 0
                 ;ZF = 1 if MSB of AL = 0
                 ;AL = 01010001 (unchanged)
                 ;PF = 0 , SF = 0
                 ;ZF = 1 because ANDing produced
                 ;  is 00
```

Rotate instructions are:

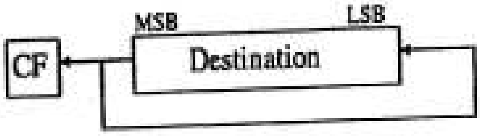
- Rotate Left (ROL).
- Rotate Right (ROR).
- Rotate Left through carry (RCL).
- Rotate Right through carry (RCR).

Format: RCR/RCL/ROR/ROL D , COUNT.

	DESTINATION	COUNT
1	Reg	1
2	Reg	1
3	Mem	CL
4	Mem	CL

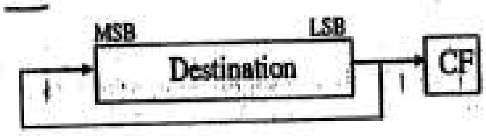
❑ ROL *Destination, Count*

- Left shift destination bits; the number of bits shifted is given by operand Count
- The MSB of the destination is shifted into CF, it also goes to the LSB of the destination
- Operand Count can be either an immediate data or register CL
- Destination can be a register or a memory location
- It modifies flags: CF OF
- OF=0 if the first operand keeps orginal sign



❑ ROR *Destination, Count*

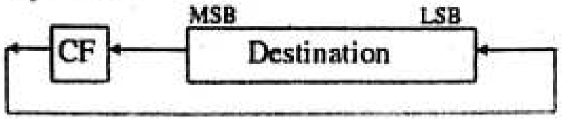
- Right shift destination bits; the number of bits shifted is given by operand Count
- The LSB of the destination is shifted into CF, it also goes to the MSB of the destination
- Operand Count can be either an immediate data or register CL
- Destination can be a register or a memory location
- It modifies flags: CF OF
- OF=0 —



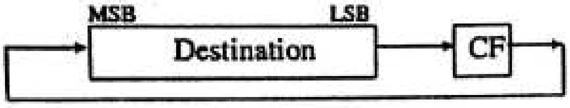
45

Scanned by CamScanner

- RCL *Destination, Count*
  - — Left shift destination bits; the number of bits shifted is given by operand Count
  - — The MSB of the destination is shifted into CF; the old CF value goes to the LSB of the destination
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF OF PF SF ZF

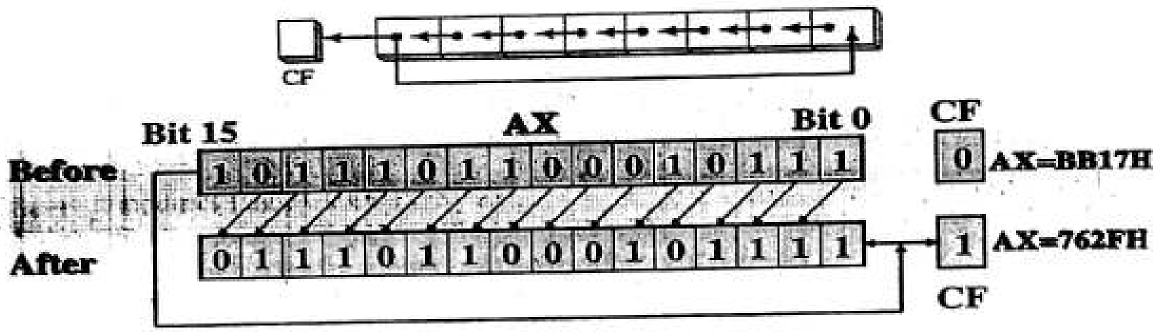


- RCR *Destination, Count*
  - — Right shift destination bits; the number of bits shifted is given by operand Count
  - — The LSB of the destination is shifted into CF, the old CF value goes to the MSB of the destination
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF OF PF SF ZF



❖ **Example:**

MOV AX, BB17H ; AX=1011101100010111b
ROL AX, 1 ; AX=0111011000101111b, CF=1



❖ **Example: In the following instructions, AL is rotated once to the left. MSB is transferred to LSB and also to Carry Flag. :**

MOV AL, 40H ; AL=01000000b, (64d)
ROL AL, 1 ; AL=10000000b, (128d), CF=0
OF = 1

Before: AL=01000000b (64 decimal, 40H)
After: AL=10000000b (128 decimal, 80H) , CF=0

46

- **ROL Des, Count:**
- **Exchanging Groups of Bits: You can use ROL to exchange the upper (bits 4–7) and lower (bits 0–3) halves of a byte.**
- **Example: In the following instructions, AL is rotated 4 times to the left. For example, 26H rotated four its in either direction becomes 62H:**

```
MOV  AL, 26H     ; AL=00100110b
ROL  AL, 4       ; AL=01100010b, CF=0
```

Before: AL=00100110b (26H)
After: AL=01100010b (62H) , CF=0

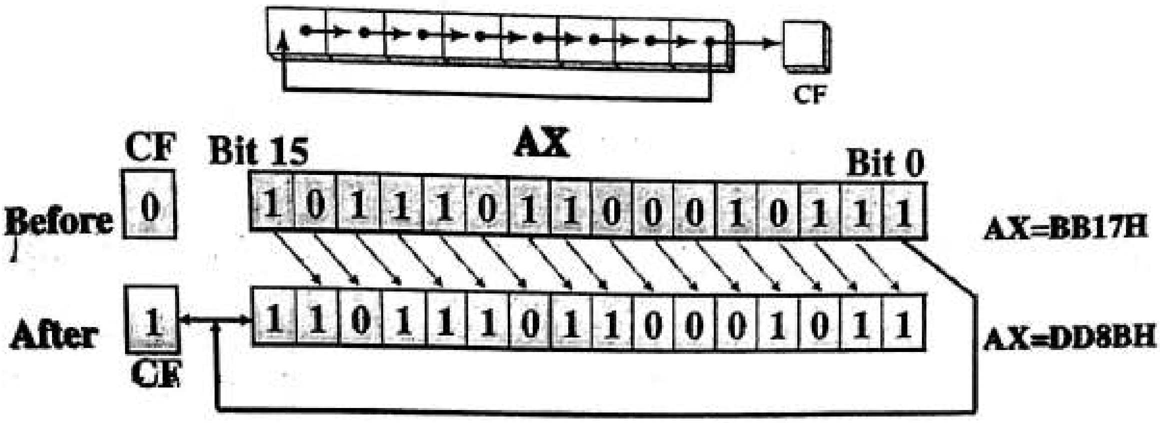
- **ROL Des, Count:**
- **Multiple Rotations: When rotating a multibyte integer by 4 bits, the effect is to rotate each hexadecimal digit one position to the right or left.**
- **Example: In the following instructions, we repeatedly rotate 6A4BH left 4 bits, eventually ending up with the original value:**

```
MOV  AX, 6A4BH   ; AX=0110101001001011b
ROL  AX, 4       ; AX=A4B6H, CF=0   (6H=0110b)
ROL  AX, 4       ; AX=4B6AH, CF=0   (AH=1010b)
ROL  AX, 4       ; AX=B6A4H, CF=0   (4H=0100b)
ROL  AX, 4       ; AX=6A4BH, CF=1   (BH=1011b)
```

❖ **<u>ROR Des, Count:</u>**

❖ **Example:**

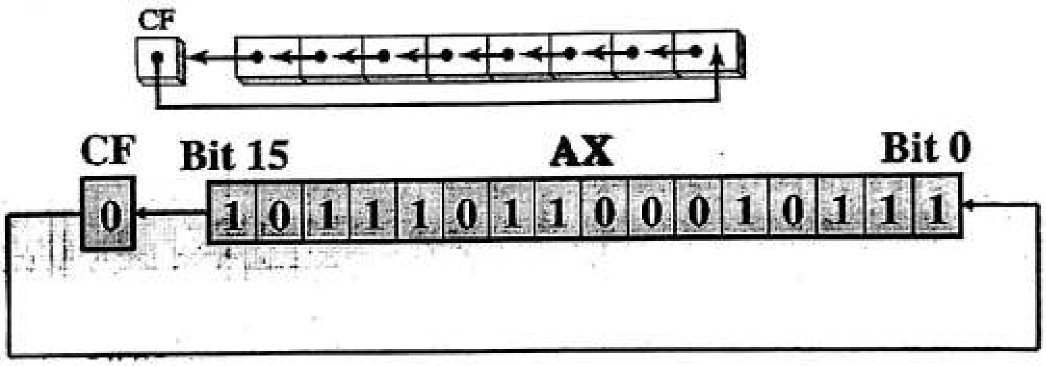
```
MOV  AX, BB17H     ; AX=1011101100010111b
ROR  AX, 1         ; AX=1101110110001011b, CF=1
```



❖ **<u>RCL Des, Count:</u>**

❖ **Example:**

```
CLC                ; CF=0
MOV  AX, BB17H     ; AX=1011101100010111b, CF=0
RCL  AX, 1         ; AX=0111011000101110b, CF=1
```



Before: AX=BB17H **After: AX=762EH**

48

EXAMPLE

What is the result in BX and CF after execution of the following instructions?

RCR BX, CL

Assume that, prior to execution of the instruction, $(CL)=04_{16}$, $(BX)=1234_{16}$, and $(CF)=0$

Solution:

The original contents of BX are

$(BX) = 0001001000110100_2 = 1234_{16}$

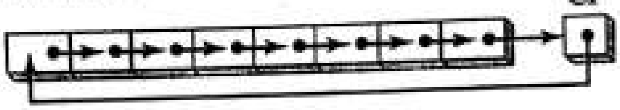
Execution of the RCR command causes a 4-bit rotate right through carry to take place on the data in BX, the results are

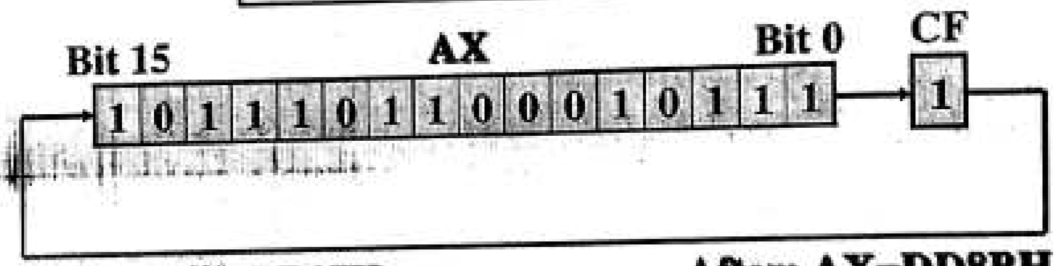
$(BX) = 1000000100100011_2 = 8123_{16}$

$(CF) = 0_2$

- **RCR Des, Count:**
- **Example:**

```
STC                ; CF=1
MOV  AX, BB17H     ; AX=1011101100010111b, CF=1
RCR  AX, 1         ; AX=1101110110001011b, CF=1
```





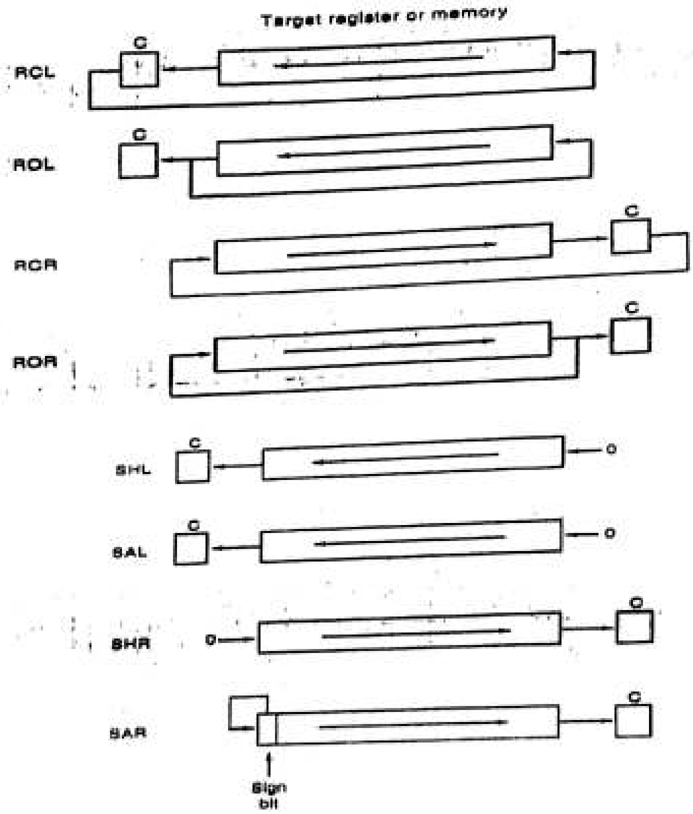
Before: AX=BB17H **After: AX=DD8BH**

```
RCR    BX, 1  ;Word in BX is rotated by 1 bit towards
              ;right and CF will contain MSB bit and
              ;LSB contain CF bit .

              ;CF = 1, BL = 00111000
RCR    BL, 1  ;Result: BL = 10011100, CF =0
              ;OF = 1  because MSB is changed to 1.

ROL    AX, 1  ;Word in AX is moved to left by 1 bit
              ;and MSB bit is to LSB, and CF

              ;CF =0 ,BH =10101110
ROL    BH, 1  ;Result: CF ,Of =1 , BH = 01011101
```



50

```
Ex:  MOV  AL,36H        ;AL=0011 0110
     ROR  AL,1          ;AL=0001 1011    CF=0
     ROR  AL,1          ;AL=1000 1101    CF=1
     ROR  AL,1          ;AL=1100 0110    CF=1
```

or:

```
     MOV  AL,36H        ;AL=0011 0110
     MOV  CL,3          ;CL=3 number of times to rotate
     ROR  AL,CL         ;AL=1100 0110    CF=1
```

```
Ex:  MOV  BX,C7E5H      ;BX=1100 0111 1110 0101
     MOV  CL,6          ;CL=6 number of times to rotate
     ROR  BX,CL         ;BX=1001 0111 0001 1111 CF=1
```

```
Ex:  MOV  AL,47H        ;AL=0100 0111
     ROL  AL,1          ;AL=1000 1110    CF=0
     ROL  AL,1          ;AL=0001 1101    CF=1
     ROL  AL,1          ;AL=0011 1010    CF=0
     ROL  AL,1          ;AL=0111 0100    CF=0
```

or:

```
     MOV  AL,47H        ;AL=0100 0111
     MOV  CL,4          ;CL=4 number of times to rotate
     ROR  AL,CL         ;BH=0111 0100    CF=0
```

Ex: Write a program that finds the number of 1s in a byte.

From the data segment:

```
DATA1    DB   97H
COUNT    DB   ?
```

From the code segment:

```
         SUB  BL,BL        ;clear BL to keep number of 1s
         MOV  DL,8         ;rotate total of 8 times
         MOV  AL,DATA1
AGAIN:   ROL  AL,1         ;rotate it once
         JNC  NEXT         ;check for 1
         INC  BL           ;if CF =1 than increment count
NEXT:    DEC  DL           ;go through this 8 times
         JNZ  AGAIN        ;if not finished go back
         MOV  COUNT,BL     save the number of ones
```


51

Ex: Write a program that finds the number of 1s in a word. Provide the count in BCD.

From the data segment:

```
DATAW1  DW   97F4H
COUNT2  DB   ?
```

From the code segment:

```
        SUB  AL,AL          ;clear BL to keep number of 1s
        MOV  DL,16          ;rotate total of 16 times
        MOV  BX,DATAW1
AGAIN:  ROL  BX,1           ;rotate it once
        JNC  NEXT           ;check for 1
        ADD  AL,1           ;if CF =1 than add 1 to count
        DAA                 ;adjust the count for BCD
NEXT:   DEC  DL             ;go through this 8 times
        JNZ  AGAIN          ;if not finished go back
        MOV  COUNT2,AL      save the number of ones
```

Note: AL had to be used to make the BCD counter because DAA instruction works only on AL.

```
Ex:     CLC                 ;clear carry, make CF=0
        MOV  AL,26H         ;AL=0010 0110
        RCR  AL,1           ;AL=0001 0011        CF=0
        RCR  AL,1           ;AL=0000 1001        CF=1
        RCR  AL,1           ;AL=1000 0100        CF=1
or:
        CLC                 ;clear carry, make CF=0
        MOV  AL,26H         ;AL=0010 0110
        MOV  CL,3           ;CL=3 number of times to rotate
        RCR  AL,CL          ;AL=1000 0100        CF=1

Ex:     STC                 ;set carry, make CF=1
        MOV  BX,37F1H       ;BX=0011 0111 1111 0001    CF=1
        MOV  CL=5           ;CL=5 number of times to rotate
        RCR  BX,CL          ;BX=0001 1001 1011 1111    CF=1

Ex:     STC                 ;set carry, make CF=1
        MOV  BL,15H         ;BL=0001 0101        CF=1
        RCL  BL,1           ;BL=0010 1011        CF=0
        RCL  BL,1           ;BL=0101 0110        CF=0
or:
        STC                 ;set carry, make CF=1
        MOV  BL,15H         ;BL=0001 0101        CF=1
        MOV  CL,2           ;CL=2 number of times to rotate
        RCL  BL,CL          ;BL=0010 1011        CF=0

Ex:     CLC                 ;clear carry, make CF=0
        MOV  AX,191CH       ;BX=0001 1001 0001 1100    CF=0
        MOV  CL=5           ;CL=5 number of times to rotate
        RCL  AX,CL          ;AX=0010 0011 1000 0001    CF=1
```


52

HW #5

1. Select an ADD instruction that will:
   (a) add BX to AX
   (b) add 12H to AL
   (c) add 22H to CX
   (d) add the data addressed by SI to AL

2. Develop a short sequence of instructions that add AL, BL, CL, DL, and AH. Save the sum in the DH register.

3. If DL = 0F3H and BH = 72H, list the difference after BH subtracts from DL and show the contents of the flag register bits.

4. Write a sequence of instructions that cube the 8-bit number found in DL. Load DL with a 5 initially and make sure that your result is a 16-bit number.

5. Develop a sequence of instructions that adds the 8-digit BCD number in AX and BX to the 8-digit BCD number in CX and DX. (AX and CX are the most-significant registers. The result must be found in CX and DX after the addition.)

6. Develop a short sequence of instructions that set (1) the three leftmost bits of DH without changing the remainder DH and store the result in BH.

7. Develop a sequence of instructions that clear (0) the rightmost four bits of AX, set (1) the leftmost three bits of AX, and invert bits 7, 8, and 9 of AX.